DYNAMIC ORTHOGONAL RANGE SEARCHING ON THE RAM, REVISITED *

Timothy M. Chan,[†] *Konstantinos Tsakalidis*[‡]

Abstract. We study a longstanding problem in computational geometry: 2-d dynamic orthogonal range reporting. We present a new data structure achieving $O\left(\frac{\log n}{\log\log n}+k\right)$ optimal query time (amortized) and $O\left(\log^{2/3+o(1)} n\right)$ update time (amortized) in the word RAM model, where n is the number of data points and k is the output size. This is the first improvement in over 10 years of Mortensen's previous result [*SIAM J. Comput.*, 2006], which has $O\left(\log^{7/8+\varepsilon} n\right)$ update time for an arbitrarily small constant $\varepsilon > 0$.

In the case of 3-sided queries, our update time reduces to $O\left(\log^{1/2+\varepsilon} n\right)$, improving Wilkinson's previous bound [ESA 2014] of $O\left(\log^{2/3+\varepsilon} n\right)$. We also obtain an improved result in higher dimensions $d \geq 3$.

1 Introduction

Orthogonal range searching is one of the most well-studied and fundamental problems in computational geometry: the goal is to design a data structure to store a set of n points so that we can quickly report all points inside a query axis-aligned rectangle. In the "emptiness" version of the problem, we just want to decide if the rectangle contains any point. (We will not study the counting version of the problem here.)

The static 2-d problem has been extensively investigated [18, 7, 28, 12, 25, 1, 24], with the current best results in the word RAM model given by Chan, Larsen, and Pătraşcu [9] for the general case (or Fries et al. [15] for the special case of 3-sided query rectangles).

In this paper, we are interested in the *dynamic* 2-d problem, allowing insertions and deletions of points. A straightforward dynamization of the standard *range tree* [30] supports queries in $O\left(\log^2 n + k\right)$ time and updates in $O\left(\log^2 n\right)$ time, where k denotes the number of reported points (for the emptiness problem, we can take $k = 0$). Mehlhorn and Näher [20] improved the query time to $O\left(\log n \log\log n + k\right)$ and the update time to $O\left(\log n \log\log n\right)$ by *dynamic fractional cascading*.

*Part of this work was done while the first author was at the University of Waterloo, Canada, and while the second author was at New York University, USA. Work of the first author was partially supported by an NSERC Discovery Grant and NSF grant CCF-1814026. Work of the second author was partially supported by NSF grants CCF-1319648 and CCF-1533564. A preliminary version of this paper appeared in *Proceedings of the 33rd Annual Symposium on Computational Geometry*, pages 28:1–28:13, 2017.

[†]*Department of Computer Science, University of Illinois at Urbana-Champaign*, tmc@illinois.edu

[‡]*Department of Computer Science, University of Liverpool*, K.Tsakalidis@liverpool.ac.uk

The first data structure to achieve logarithmic query and update (amortized) time was presented by Mortensen [22]. In fact, he obtained *sub*logarithmic bounds in the word RAM model: the query time is $O\left(\frac{\log n}{\log\log n}+k\right)$ and the amortized update time is $O\left(\log^{7/8+\varepsilon} n\right)$ where ε denotes an arbitrarily small positive constant.

On the lower bound side, Alstrup et al. [2] showed that any data structure with t_u update time for 2-d range emptiness requires $\Omega\left(\frac{\log n}{\log(t_u \log n)}\right)$ query time in the cell-probe model. Thus, Mortensen's query bound is optimal for any data structure with polylogarithmic update time. However, it is conceivable that the update time could be improved further while keeping the same query time. Indeed, the $O\left(\log^{7/8+\varepsilon} n\right)$ update bound looks too peculiar to be optimal, one would think.

Let us remark how intriguing this type of "fractional-power-of-log" bound is, which showed up only on a few occasions in the literature. For example, Chan and Pătraşcu [10] gave a dynamic data structure for 1-d rank queries (counting number of elements less than a given value) with $O\left(\frac{\log n}{\log\log n}\right)$ query time and $O\left(\log^{1/2+\varepsilon} n\right)$ update time. Chan and Pătraşcu also obtained more $\sqrt{\log n}$-type results for various offline range counting problems. Another example is Wilkinson's recent paper [27]: he studied a special case of 2-d orthogonal range reporting for *2-sided* and *3-sided* rectangles and obtained a solution with $O\left(\frac{\log n}{\log\log n}+k\right)$ amortized query time, $O\left(\log^{1/2+\varepsilon} n\right)$ update time for the 2-sided case, and $O\left(\log^{2/3+\varepsilon} n\right)$ update time for 3-sided; the latter improves Mortensen's $O\left(\log^{5/6+\varepsilon} n\right)$ update bound for 3-sided [22]. He also showed that in the insertion-only and deletion-only settings, it is possible to get fractional-power-of-log bounds for both the update and the query time. However, he was unable to make progress for general 4-sided rectangles in the insertion-only and deletion-only settings, let alone the fully dynamic setting.

New results. Our main new result is a fully dynamic data structure for 2-d orthogonal range reporting with $O\left(\frac{\log n}{\log\log n}+k\right)$ optimal query time and $O\left(\log^{2/3+o(1)} n\right)$ update time, greatly improving Mortensen's $O\left(\log^{7/8+\varepsilon} n\right)$ bound. In the 3-sided case, we obtain $O\left(\log^{1/2+\varepsilon} n\right)$ update time, improving Wilkinson's $O\left(\log^{2/3+\varepsilon} n\right)$ bound. (See Table 1 for comparison.) Our update bounds seem to reach a natural limit with this type of approach. In particular, it is not unreasonable to conjecture that the near-$\sqrt{\log n}$ update bound for the 3-sided case is close to optimal, considering prior "fractional-power-of-log" upper-bound results in the literature (though there have been no known lower bounds of this type so far).

Like previous methods, our bounds are amortized (this includes query time). Our results are in the word-RAM model, under the standard assumption that the word size w is at least $\log n$ bits (in fact, except for an initial predecessor search during each query/update, we only need operations on $(\log n)$-bit words). Even to researchers uncomfortable with sublogarithmic algorithms on the word RAM, such techniques are still relevant. For example, Mortensen extended his data structure to $d \geq 3$ dimensions and obtained $O\left(\left(\frac{\log n}{\log\log n}\right)^{d-1}+k\right)$ query time and $O\left(\log^{d-9/8+\varepsilon} n\right)$ update time, even in the real-RAM model (where each

Table 1: Dynamic planar orthogonal range reporting: previous and new results.

		Update time	Query time
4-sided	Lueker and Willard [30]	$\log^2 n$	$\log^2 n + k$
	Mehlhorn and Näher [20]	$\log n \log\log n$	$\log n \log\log n + k$
	Mortensen [22]	$\log^{7/8+\varepsilon} n$	$\frac{\log n}{\log\log n} + k$
	New	$\log^{2/3} n \log^{O(1)} \log n$	$\frac{\log n}{\log\log n} + k$
3-sided	McCreight [19]	$\log n$	$\log n + k$
	Willard [29]	$\frac{\log n}{\log\log n}$	$\frac{\log n}{\log\log n} + k$
	Mortensen [22]	$\log^{5/6+\varepsilon} n$	$\frac{\log n}{\log\log n} + k$
	Wilkinson [27]	$(\log n \log\log n)^{2/3}$	$\log n + k$
	Wilkinson [27]	$\log^{2/3+\varepsilon} n$	$\frac{\log n}{\log\log n} + k$
	New	$\log^{1/2+\varepsilon} n$	$\frac{\log n}{\log\log n} + k$

word can hold an input real number or a $(\log n)$-bit number). We can also obtain further improvements, with the same query time and $O\left(\log^{d-2+O(1/d)} n\right)$ update time.

Overview of techniques: Micro- and macro-structures. Our solution builds on ideas from Mortensen's paper [22]. His paper was long and not easy to follow, unfortunately; we strive for a clearer organization and a more accessible exposition (which in itself would be a valuable contribution).

The general strategy towards obtaining fractional-power-of-log bounds, in our view, can be broken into two parts: the design of what we will call *micro-structures* and *macro-structures*.

- Micro-structures refer to data structures for handling a small number s of points; by "small", we mean $s = 2^{\log^\alpha n}$ for some fraction $\alpha < 1$ (rather than s being polylogarithmic, as is more usual in other contexts). When s is small, by *rank space reduction* we can make the universe size small, and as a consequence are able to pack multiple points (about $\frac{w}{\log s}$) into a single word. As observed by Chan and Pătraşcu [10] and by Wilkinson [27], we can design micro-structures by thinking of each word as a block of multiple points, and borrowing known techniques from the world of *external-memory* algorithms (specifically, *buffer trees* [4]) to achieve *(sub)constant* amortized update time. Alternatively, Mortensen described his micro-structures from scratch, which required a more complicated solution to a certain "pebble game" [22, Section 6].

  One subtle issue is that to simulate rank space reduction dynamically, we need *list labeling* techniques, which, if not carefully implemented, can worsen the exponent in the update bound (as was the case in both Mortensen's and Wilkinson's solutions).

- Macro-structures refer to data structures for large input size n, constructed using micro-structures as black boxes. This part does not involve bit packing, and relies on more traditional geometric divide-and-conquer techniques such as higher-degree

range trees, as in Mortensen's and in Chan and Pătraşcu's solutions, with degree $2^{\log^{\beta} n}$ for some fraction $\beta < 1$. Van Emde Boas recursion is also a crucial ingredient in Mortensen's macro-structures.

Our solution will require a number of new ideas in both micro- and macro-structures. On the micro level, we bypass the "pebbling" problem by explicitly invoking external-memory techniques, as in Wilkinson's work [27], but we handle the list labeling issue more carefully in order to avoid worsening the update time. On the macro level, we use higher-degree range trees but with a more intricate analysis (involving Harmonic series, interestingly), plus a few bootstrapping steps, in order to achieve the best update and query bounds.

2 Preliminaries

In all our algorithms, we assume that during each query or update operation, we are given a pointer to the predecessor/successor of the x- and y-values of the given point or rectangle. At the end, we can add the cost of predecessor search to the query and update time (which is no bigger than $O\left(\sqrt{\log n}\right)$ [3] in the word RAM model).

We assume a word RAM model that allows for a constant number of "non-standard" operations on w-bit words. By setting $w := \delta \log n$ for a sufficiently small constant δ, these operations can be simulated in constant time by table lookup, after preprocessing the tables in $2^{O(w)} = n^{O(\delta)}$ time.

For simplicity, we concentrate on emptiness queries; all our algorithms can be modified for reporting queries, with an additional $O(k)$ term to the query time bounds.

A *3-sided* query deals with a rectangle that is unbounded on the left or right side. In contrast, a *flipped 3-sided* query deals with a rectangle that is unbounded on the top or bottom side. (A flipped 4-sided query is the same as a 4-sided query.) A *2-sided* (or *dominance*) query deals with a rectangle that is unbounded on two adjacent sides.

Let $[n]$ denote $\{0, 1, \ldots, n-1\}$.

We now quickly review a few useful tools.

List labeling. *Monotone list labeling* is the problem of assigning *labels* to a dynamic set of totally ordered elements, such that whenever $x < y$, the label of element x is less than the label of element y. As elements are inserted, we are allowed to change labels. The following result is well known:

Lemma 1. *[13] (see also [14, 6, 16]) A monotone labeling for n totally ordered elements with labels in $\left[n^{O(1)}\right]$ can be maintained under insertions by making $O(n \log n)$ label changes in total, in $O(n \log n)$ total time.*

Weight-balancing. *Weight-balanced B-trees* [5] are B-tree implementations with a rebalancing scheme that is based on the nodes' *weights*, i.e., subtree sizes, in order to support

updates of secondary structures efficiently.

Lemma 2. *[5, Lemma 4] In a weight-balanced B-tree of degree s, nodes at height i have weight $\Theta(s^i)$, and any sequence of n insertions requires at most $O(n/s^i)$ splits of nodes at height i.*

Colored predecessors. *Colored predecessor searching* is the problem of maintaining a dynamic set of multi-colored, totally ordered elements and searching for the predecessors with a given color.

Lemma 3. *[22, Theorem 14] Colored predecessor searches and updates on n colored, totally ordered elements can be supported in $O\left(\log^2 \log n\right)$ time deterministically.*

Van Emde Boas transformation. A crucial ingredient we will use is a general technique of Mortensen [21, 22] that transforms any given data structure for orthogonal range emptiness on small sets of $s^{O(1)}$ points, to one for point sets in a *narrow grid* $[s] \times \mathbb{R}$, at the expense of an increase in cost by $\log \log n$ factors. We state the result in a slightly more general form, allowing the narrow grid to be $X \times \mathbb{R}$ for an arbitrary set X of $O(s)$ values:

Lemma 4. *[22, Theorem 1] Let X be a set of $O(s)$ values. Given a dynamic data structure for j-sided orthogonal range emptiness ($j \in \{3, 4\}$) on s^2 points in $X \times \mathbb{R}$ with (amortized) update time $U_j(s, s^2)$ and query time $Q_j(s, s^2)$, there exists a dynamic data structure for j-sided orthogonal range emptiness on n points in $X \times \mathbb{R}$ with update time $U_j(s, n) = O\left(U_j(s, s^2) \log^2 \log n\right)$ and query time $Q_j(s, n) = O\left(Q_j(s, s^2) \log \log n\right)$.*

If the given data structure supports updates to X (i.e., insertions/deletions of values in X) in $U_X(s)$ time and this update procedure depends solely on X (and not the point set), the new data structure can support updates to X in $U_X(s)$ time.

Mortensen's transformation is obtained via a van-Emde-Boas-like recursion [26]. His paper stated the above lemma only for the case of a static y-universe (there, one of the $\log \log$-factors in the update time can be eliminated). It isn't entirely clear to us how he dealt with the issue of dynamic y-universes. For the sake of completeness, we give a concise re-description of the proof in the Appendix, to show how the data structure can handle the dynamic y-universe setting.

3 Part 1: Micro-Structures

We first design *micro-structures* for 3- and 4-sided dynamic orthogonal range emptiness when the number of points s is small. This part heavily relies on bit-packing techniques.

3.1 Static universe

We begin with the case of a static universe $\left[s^{O(1)}\right]^2$.

Lemma 5. *For s points in the static universe $\left[s^{O(1)}\right]^2$, there exist data structures for dynamic orthogonal range emptiness that support*

(i) updates in $O\left(\frac{\log^2 s}{w}+1\right)$ amortized time and 3-sided queries in $O(\log s)$ amortized time;

(ii) updates in $O\left(\frac{\log^3 s}{w}+1\right)$ amortized time and 4-sided queries in $O\left(\log^2 s\right)$ amortized time.

Proof. We mimick existing *external-memory* data structures with a block size of $B := \left\lceil \frac{\delta w}{\log s} \right\rceil$ for a sufficiently small constant δ, observing that B points can be packed into a single word.

(i) For the 3-sided case, Wilkinson [27, Lemma 1] has already adapted such an external-memory data structure, namely, a *buffered* version of a binary *priority search tree* due to Kumar and Schwabe [17] (see also Brodal's more recent work [8]), which is similar to the *buffer tree* of Arge [4]. For 3-sided rectangles unbounded to the left/right, the priority search tree is ordered by y, where each node stores $O(B)$ x-values. Wilkinson obtained $O\left(\frac{1}{B}\cdot\log s+1\right) = O\left(\frac{\log^2 s}{w}+1\right)$ amortized update time and $O(\log s)$ amortized query time.

(ii) For the general 4-sided case, we use a *buffered* version of a binary *range tree*. Although we are not aware of prior work explicitly giving such a variant of the range tree, the modifications are straightforward, and we will provide only a rough outline. The range tree is ordered by y. Each node holds a buffer of up to B update requests that have not yet been processed. Each node is also augmented with a 1-d binary *buffer tree* (already described by Arge [4]) for the x-projection of the points. To insert or delete a point, we add the update request to the root's buffer. Whenever a buffer's size of a node exceeds B, we empty the buffer by applying the following procedure: we divide the list of $\Theta(B)$ update requests into two sublists for the two children in $O(1)$ time using a non-standard word operation (since B update requests fit in a word); we then pass these sublists to the buffers at the two children, and also pass another copy of the list to the node's 1-d buffer tree. These 1-d updates cost $O\left(\frac{1}{B}\cdot\log s\right)$ each [4], when amortized over $\Omega(B)$ updates. Since each update eventually travels to $O(\log s)$ nodes of the range tree, the amortized update time of the 4-sided structure is $O\left(\frac{1}{B}\log^2 s+1\right) = O\left(\frac{\log^3 s}{w}+1\right)$.

A 4-sided query is answered by following two paths in the range tree in a top-down manner, performing $O(\log s)$ 1-d queries; since each 1-d query takes $O(\log s)$ time, the overall query time is $O\left(\log^2 s\right)$. However, before we can answer the query, we need to first empty the buffers along the two paths of the range tree. This can be done by applying the procedure in the preceding paragraph at the $O(\log s)$ nodes top-down; this takes $O(\log s)$ time, plus the time needed for $O(B\log s)$ 1-d updates, costing $O\left(\frac{1}{B}\cdot\log s\right)$ each [4]. The final amortized query time is thus $O\left(\log^2 s\right)$. □

The above methods can be modified for range reporting with an extra query cost of $O(k)$ for reporting k output points.

Notice that the above update time is *constant* when the number of points s is as large as $2^{\sqrt{w}}$ for 3-sided queries or $2^{w^{1/3}}$ for 4-sided.

(It is possible to eliminate one of the logarithmic factors in the query time for the above 4-sided result, by augmenting nodes of the range tree with 3-sided structures. However, this alternative causes difficulty later in the extension to dynamic universes. Besides, the larger query time turns out not to matter for our macro-structures at the end.)

3.2 Dynamic universe

To make the preceding data structures support a dynamic universe, the simplest way is to apply monotone list labeling (Lemma 1), which maps coordinates to $\left[s^{O(1)}\right]^2$. Whenever a label of a point changes, we just delete the point and reinsert a copy with the new coordinates into the data structure. However, since the total number of label changes is $O(s \log s)$ over s insertions, this slows down the amortized update time by a $\log s$ factor and will hurt the final update bound.

Our approach is as follows. We first observe that the list labeling approach works fine for changes to the y-universe. For changes to the x-universe, we switch to a "brute-force" method with large running time. This turns out to be adequate for our macro-structures at the end, since the number of x-universe changes will be relatively small, as we will see later in Section 4.1. (The brute-force idea can also be found in Mortensen's paper [22], but his macro-structures were less efficient.)

Lemma 6. *Both data structures in Lemma 5 can be modified to work for s points in a universe $X \times Y$ with $|X|, |Y| = O(s)$. The update and query time bounds are the same, and we can support*

(a) updates to Y in $O\left(\log^2 \log s\right)$ amortized time (given a pointer to the predecessor/successor in Y), and

(b) updates to X in $2^{O(w)}$ time, where the update procedure for X depends solely on X (and not the point set).

Proof. (a) To start, let us assume that $X = \left[s^{O(1)}\right]$ but Y is arbitrary. We divide the sorted list Y into $O(s/A)$ blocks of size $\Theta(A)$ for a parameter A to be set later. It is easy to maintain such a blocking using $O(s/A)$ number of block merges and splits over s updates. (Such a blocking was also used by Wilkinson [27].) We maintain a monotone labeling of the blocks by Lemma 1. In the proof of Lemma 5(i) or (ii), we construct the y-ordered priority search tree or range tree using the block labels as the y-values. Each leaf then corresponds to a block. We build a small range tree for each leaf block to support updates and queries for the $O(A)$ points in, say, $O\left(\log^2 A\right)$ time. We can encode a y-value $\eta \in Y$ by a pair consisting of the label of the block containing η (from $[O(s/A)]$), and the rank of η with respect to the block (from $[O(A)]$). We will use these encoded values, which still are $O(\log s)$-bit long, in all the buffers. The block labels provide sufficient information to pass the update requests to the leaves and the x-ordered 1-d buffer trees. For a particular leaf,

the ranks with respect to its corresponding block provide sufficient information to handle a query or update at this leaf.

During each block split/merge and each block label change, we need to first empty the buffers along the path to the block before applying the change. This can be done by applying the procedure from the proof of Lemma 5 at $O(\log s)$ nodes top-down, requiring $O(\log s)$ amortized time. Since the total number of block label changes is $O\left(\frac{s}{A}\log\frac{s}{A}\right)$, the total time for these steps is $O\left(\frac{s}{A}\log\frac{s}{A}\cdot\log s\right) = O(s)$ by setting $A := \log^2 s$. The amortized cost for these steps is thus $O(1)$. The final amortized cost is $O\left(\log^2 A\right) = O\left(\log^2\log s\right)$.

(b) Now, we remove the $X = \left[s^{O(1)}\right]$ assumption. We assign elements in X to labels in $[O(s)]$, but this time we do not use monotone labeling. This way, the label of an x-value does not need to change once it is assigned. Buffers store the labels rather than the actual x-values. However, the non-standard word operations on the x-values in the buffers have to be done differently. For example, consider the operation of finding the minimum of B x-values packed in a word (needed to implement the buffered priority search tree); in the modified operation, we are given B labels packed in a word and want to output the minimum of the B x-values corresponding to these labels. Such an operation can still be simulated by table lookup, where the answers to all $2^{O(w)}$ possible inputs can be precomputed in $2^{O(w)}$ time. Inserting a new x-value to X requires more work now: during an insertion of X, after we assign the new x-value a new label in $[O(s)]$, we need to compute $2^{O(w)}$ table entries from scratch by brute force, taking $2^{O(w)}$ time. □

4 Part 2: Macro-Structures

We now present macro-structures for 3- and 4-sided dynamic orthogonal range emptiness when the number of points n is large, by using micro-structures as black boxes. This part does not involve bit packing (and hence is more friendly to computational geometers). The transformation from micro- to macro-structures is based on variants of range trees.

4.1 Range tree transformation I

We present our first transformation. As warm up, we start by stating a shorter version of the transformation, which is easier to understand (this simpler version is sufficient in the special case when there are no updates to the X universe). We then state and prove the long version that we will actually use.

Lemma 7. (Abridged version) *Given a data structure $\mathcal{D}_j$ for dynamic j-sided orthogonal range emptiness ($j \in \{3,4\}$) on n points in $X \times \mathbb{R}$ ($|X| = O(s)$) with (amortized) update time $U_j(s,n)$ and query time $Q_j(s,n)$, where updates to X are allowed with no extra cost, there exist data structures for dynamic orthogonal range emptiness on n points in the plane with the following amortized update and query time:*

(i) for the 3-sided case,

$$\begin{aligned} U_3(n) &= O\left(U_3(s,n)\log_s n + \log_s n \log^2\log n\right) \\ Q_3(n) &= O\left(Q_3(s,n)\log_s n + \log_s n \log^2\log n\right); \end{aligned}$$

(ii) for the 4-sided case,

$$\begin{aligned} U_4(n) &= O\left(\left(U_4(s,n)+U_3(s,n)\right)\log_s n + \log_s n \log^2\log n\right) \\ Q_4(n) &= O\left(Q_4(s,n) + Q_3(s,n)\log_s n + \log_s n \log^2\log n\right). \end{aligned}$$

Lemma 7. (Long version) *Given a family of data structures $\mathcal{D}_j^{(i)}$ $(i \in \{1,\ldots,\log_s n\})$ for dynamic j-sided orthogonal range emptiness $(j \in \{3,4\})$ on n points in $X \times \mathbb{R}$ $(|X| = O(s))$ with (amortized) update time $U_j^{(i)}(s,n)$ and query time $Q_j^{(i)}(s,n)$, where updates to X take $U_X^{(i)}(s)$ time, there exist data structures for dynamic orthogonal range emptiness on n points in the plane with the following amortized update and query time:*

(i) for the 3-sided case,

$$\begin{aligned} U_3(n) &= O\left(\sum_{i=1}^{\log_s n} U_3^{(i)}(s,n) + \sum_{i=1}^{\log_s n} \frac{U_X^{(i)}(s)}{s^{i-1}} + \log_s n \log^2\log n\right) \\ Q_3(n) &= O\left(\max_i Q_3^{(i)}(s,n)\log_s n + \log_s n \log^2\log n\right); \end{aligned}$$

(ii) for the 4-sided case,

$$\begin{aligned} U_4(n) &= O\left(\sum_{i=1}^{\log_s n} (U_4^{(i)}(s,n) + U_3^{(i)}(s,n)) + \sum_{i=1}^{\log_s n} \frac{U_X^{(i)}(s)}{s^{i-1}} + \log_s n \log^2\log n\right) \\ Q_4(n) &= O\left(\max_i Q_4^{(i)}(s,n) + \max_i Q_3^{(i)}(s,n)\log_s n + \log_s n \log^2\log n\right). \end{aligned}$$

Proof. We store a range tree ordered by x, implemented as a degree-s weight-balanced B-tree. (Deletions can be handled lazily without changing the weight-balanced tree; we can rebuild periodically when n decreases or increases by a constant factor.) At every internal node v at height i, we let X_v be the set of x-coordinates of the $O(s)$ vertical lines dividing the children nodes of v, and store the points in its subtree in the given data structure $\mathcal{D}_j^{(i)}$ for j-sided orthogonal range emptiness on a *narrow grid* $X_v \times \mathbb{R}$, where the x-coordinate of every point is replaced with its predecessor in X_v. We also store the y-coordinates of these points in a colored predecessor searching structure of Lemma 3, where points in the same child's vertical slab are assigned the same color. And we store the x-coordinates in another colored predecessor searching structure, where X_v is colored black and the rest is colored white.

To insert or delete a point, we update the narrow-grid structures at the nodes along the path in the tree. This takes $O\left(\sum_{i=1}^{\log_s n} U_j^{(i)}(s,n)\right)$ total time. Note that given the y-predecessor/successor of the point at a node, we can obtain the y-predecessor/successor at the child by using the colored predecessor searching structure. We can also determine the x-predecessor in X_v by another colored predecessor search. The extra cost for descending along the path is thus $O\left(\log_s n \log^2 \log n\right)$.

To keep the tree balanced, we need to handle node splits. For nodes at height i, there are $O\left(n/s^i\right)$ splits by Lemma 2. Each such split requires rebuilding two narrow-grid structures on $O\left(s^i\right)$ points, which can be done naively by $O\left(s^i\right)$ insertions to empty structures. This has $O\left(\sum_{i=1}^{\log_s n} \left(n/s^i\right) \cdot s^i U_j^{(i)}(s,n)\right)$ total cost, i.e., an amortized cost of $O\left(\sum_{i=1}^{\log_s n} U_j^{(i)}(s,n)\right)$. A split of a child of v also requires updating (deleting and reinserting) the points at the child's slab. This has $O\left(\sum_{i=1}^{\log_s n} \left(n/s^{i-1}\right) \cdot s^{i-1} U_j^{(i)}(s,n)\right)$ total cost, i.e., an amortized cost of $O\left(\sum_{i=1}^{\log_s n} U_j^{(i)}(s,n)\right)$. Moreover, a split of a child of v requires an update to X_v. This has $O\left(\sum_{i=1}^{\log_s n} \left(n/s^{i-1}\right) \cdot U_X^{(i)}(s)\right)$ total cost, i.e., an amortized cost of $O\left(\sum_{i=1}^{\log_s n} \left(1/s^{i-1}\right) \cdot U_X^{(i)}(s)\right)$. Furthermore, the split requires $O\left(1\right)$ updates to the colored predecessor structures for X_v and $O\left(s\right)$ updates to the colored predecessor structures at the two new nodes. This has $O\left(\sum_{i=1}^{\log_s n} \left(n/s^{i-1}\right) \cdot \log^2 \log n + \sum_{i=1}^{\log_s n} \left(n/s^i\right) \cdot s \log^2 \log n\right) = O\left(n \log^2 \log n\right)$ total cost, i.e., an amortized cost of $O\left(\log^2 \log n\right)$.

To answer a 3-sided query, we proceed down a path of the tree and perform queries in the narrow-grid structures at nodes along the path. These queries take total time $O\left(\log_s n \cdot \max_i Q_3^{(i)}(s,n)\right)$. As before, given the y-predecessor/successor of the coordinates of the rectangle at a node, we can obtain the y-predecessor/successor at the child by using the colored predecessor searching structure. We can also determine the x-predecessor in X_v by another colored predecessor search. The extra cost for descending along the path is thus $O\left(\log_s n \log^2 \log n\right)$.

To answer a 4-sided query, we find the highest node v whose dividing vertical lines cut the query rectangle, by descending along a path from the root in $O\left(\log_s n \log^2 \log n\right)$ time. We obtain two 3-sided queries at two children of v, which can be answered as above, plus a remaining query that can be answered via the narrow-grid structure at v in $O\left(\max_i Q_4^{(i)}(s,n)\right)$ time. □

Combining with our preceding micro-structures and the van Emde Boas transformation, we obtain the following results, achieving the desired update time but slightly suboptimal query time (which we will fix later):

Theorem 1. *Given n points in the plane, there exist data structures for dynamic orthogonal range emptiness that support*

(i) updates in amortized $O\left(\log^{1/2} n \log^{O(1)} \log n\right)$ time and 3-sided queries in amortized $O\left(\log n \log \log n\right)$ time;

(ii) updates in amortized $O\left(\log^{2/3} n \log^{O(1)} \log n\right)$ *time and 4-sided queries in amortized* $O(\log n \log\log n)$ *time.*

Proof. (i) For the 3-sided case, Lemmata 5(i) and 6 give micro-structures with update time $O\left(\frac{\log^2 s}{\overline{w}} + \log^2 \log s\right)$ and query time $O(\log s)$, while supporting updates to X in $2^{O(\overline{w})}$ time. Observe that we can choose to work with a smaller word size $\overline{w} \leq w$, so long as $\overline{w} = \Omega(\log s)$. We choose $\overline{w} := \delta i \log s$ for a sufficiently small absolute constant δ and for any given $i \in [2, \log_s n]$. To summarize, we have micro-structures with the following update time, query time, cost for updating X:

$$\begin{aligned} U_3^{(i)}(s, s^2) &= O\left(\frac{\log s}{i} + \log^2 \log s\right) \\ Q_3^{(i)}(s, s^2) &= O(\log s) \\ U_X^{(i)}(s) &= s^{O(\delta i)} \end{aligned}$$

For the special case $i = 1$, we use a standard priority search tree, with $U_3^{(1)}(s, s^2), Q_3^{(1)}(s, s^2) = O(\log s)$ and $U_X^{(1)}(s) = 0$. By Lemma 4 (van Emde Boas transformation), we obtain narrow-grid structures with update time $U_3^{(i)}(s, n) = O(U_3^{(i)}(s, s^2) \log^2 \log n)$ and query time $Q_3^{(i)}(s, n) = O(Q_3^{(i)}(s, s^2) \log \log n)$. Substituting into Lemma 7, we obtain

$$\begin{aligned} U_3(n) &= O\left(\sum_{i=1}^{\log_s n} \frac{\log s \log^2 \log n}{i} \;+\; \log_s n \log^4 \log n \;+\; \sum_{i=2}^{\log_s n} \frac{s^{O(\delta i)}}{s^{i-1}}\right) \\ &= O\left(\log s \log^3 \log n \;+\; \log_s n \log^4 \log n\right), \end{aligned}$$

since the first sum is a Harmonic series and the second sum is a geometric series. (This assumes a sufficiently small constant for δ, as the hidden constant in the exponent $O(\delta i)$ does not depend on δ.) Furthermore,

$$\begin{aligned} Q_3(n) &= O\left(\log s \log_s n \log\log n \;+\; \log_s n \log^2 \log n\right) \\ &= O\left(\log n \log\log n \;+\; \log_s n \log^2 \log n\right). \end{aligned}$$

We set $s := 2^{\sqrt{\log n}}$ to get $U_3(n) = O\left(\log^{1/2} n \log^{O(1)} \log n\right)$ and $Q_3(n) = O(\log n \log\log n)$.

(ii) Similarly, for the 4-sided case, Lemmata 5(ii) and 6 with a smaller word size $\overline{w} := \delta i \log s$ give micro-structures with

$$\begin{aligned} U_4^{(i)}(s, s^2) &= O\left(\frac{\log^2 s}{i} + \log^2 \log s\right) \\ Q_4^{(i)}(s, s^2) &= O\left(\log^2 s\right) \\ U_X^{(i)}(s) &= s^{O(\delta i)}. \end{aligned}$$

For the special case $i = 1$, we use a standard range tree, achieving $U_4^{(1)}(s, s^2), Q_4^{(1)}(s, s^2) = O\left(\log^2 s\right)$ and $U_X^{(1)}(s) = 0$. Applying Lemmata 4 and 7, we obtain

$$\begin{aligned} U_4(n) &= O\left(\sum_{i=1}^{\log_s n} \frac{\log^2 s \log^2 \log n}{i} + \log_s n \log^4 \log n + \sum_{i=2}^{\log_s n} \frac{s^{O(\delta i)}}{s^{i-1}}\right) \\ &= O\left(\log^2 s \log^3 \log n + \log_s n \log^4 \log n\right) \end{aligned}$$

and

$$\begin{aligned} Q_4(n) &= O\left(\log^2 s \log\log n + \log s \log_s n \log\log n + \log_s n \log^2 \log n\right) \\ &= O\left(\log^2 s \log\log n + \log n \log\log n + \log_s n \log^2 \log n\right). \end{aligned}$$

We set $s := 2^{\log^{1/3} n}$ to obtain $U_4(n) = O\left(\log^{2/3} n \log^{O(1)} \log n\right)$ and $Q_4(n) = O(\log n \log\log n)$. □

4.2 Range tree transformation II

We now reduce the query time to optimal by another transformation:

Lemma 8. *Given a data structure $\mathcal{D}_j$ for dynamic j-sided orthogonal range emptiness ($j \in \{2, 3, 4\}$) on n points in $X \times \mathbb{R}$ ($|X| = O(s)$) with (amortized) update time $U_j(s, n)$ and query time $Q_j(s, n)$, where updates to X are allowed with no extra cost, and given a data structure for dynamic $(j-1)$-sided orthogonal range emptiness on n points with update time $U_{j-1}(n)$ and query time $Q_{j-1}(n)$, there exist data structures for dynamic flipped j-sided orthogonal range emptiness ($j \in \{3, 4\}$) on n points in the plane with the following amortized update and query time:*

$$\begin{aligned} U_j(n) &= O\left(\left(U_j(s, n) + U_{j-1}(n)\right) \log_s n + \log_s n \log^2 \log n\right) \\ Q_j(n) &= O\left(Q_j(s, n) + Q_{j-1}(n) + \log_s n \log^2 \log n\right). \end{aligned}$$

Proof. We modify the range tree in the proof of Lemma 7, where every internal node is augmented with a $(j-1)$-sided structure on the set of points in its subtree.

During an insertion or deletion of a point, we update the narrow-grid structures along a path as before, in $O(\log_s n \cdot U_j(s, n))$ time. We now also need to update the $(j-1)$-sided structures at nodes along the path. This adds $O(U_{j-1}(n) \log_s n)$ to the update time.

During rebalancing, each split of a node at height i now requires rebuilding the $(j-1)$-sided structures, which can be done naively by $O(s^i)$ insertions to an empty structure. This has $O\left(\sum_{i=1}^{\log_s n} (n/s^i) \cdot s^i U_{j-1}(n)\right)$ total cost, i.e., an amortized cost of $O(U_{j-1}(n) \log_s n)$.

To answer a flipped j-sided query, we find the highest node v whose dividing vertical lines cut the query rectangle, by descending along a path from the root as before in

$O\left(\log_s n \log^2 \log n\right)$ time. We obtain two $(j-1)$-sided queries at two children of v, plus a query in the narrow-grid structure at v. (In the case $j = 3$, it is important here that we are given a *flipped* 3-sided query.) The two $(j-1)$-sided queries can be answered directly using the augmented structures. These queries take $O\left(Q_j(s,n) + Q_{j-1}(n)\right)$ time. □

We obtain our final results by bootstrapping:

Theorem 2. *Given n points in the plane, there exist data structures for dynamic orthogonal range emptiness that support*

(i) updates in amortized $O\left(\log^{1/2+O(\varepsilon)} n\right)$ time and 3-sided queries in amortized $O\left(\frac{\log n}{\log\log n}\right)$ time for an arbitrarily small constant $\varepsilon > 0$;

(ii) updates in amortized $O\left(\log^{2/3} n \log^{O(1)} \log n\right)$ time and 4-sided queries in amortized $O\left(\frac{\log n}{\log\log n}\right)$ time.

Proof. (i) Theorem 1(i) achieves

$$\begin{aligned} U_3(s, s^2) &= O\left(\log^{1/2} s \log^{O(1)} \log s\right) \\ Q_3(s, s^2) &= O\left(\log s \log\log s\right). \end{aligned}$$

Wilkinson [27] has given a data structure for 2-sided (dominance) queries with

$$\begin{aligned} U_2(n) &= O\left(\log^{1/2+\varepsilon} n\right) \\ Q_2(n) &= O\left(\frac{\log n}{\log\log n}\right). \end{aligned}$$

Applying Lemmata 4 and 8, we obtain

$$\begin{aligned} U_3(n) &= O\left(\log^{1/2} s \log_s n \log^{O(1)} \log n \;+\; \log^{1/2+\varepsilon} n \log_s n + \log_s n \log^2 \log n\right) \\ Q_3(n) &= O\left(\log s \log\log s \log\log n \;+\; \frac{\log n}{\log\log n} \;+\; \log_s n \log^2 \log n\right). \end{aligned}$$

We set $s := 2^{\frac{\log n}{\log^3 \log n}}$ to get $U_3(n) = O\left(\log^{1/2+O(\varepsilon)} n\right)$ and $Q_3(n) = O\left(\frac{\log n}{\log\log n}\right)$.

These time bounds for flipped 3-sided queries apply to (non-flipped) 3-sided queries as well, by a symmetric data structure.

(ii) Similarly, Theorem 1(ii) achieves

$$\begin{aligned} U_4(s) &= O\left(\log^{2/3} s \log^{O(1)} \log s\right) \\ Q_4(s) &= O\left(\log s \log\log s\right). \end{aligned}$$

Part (i) above gives

$$\begin{aligned} U_3(n) &= O\left(\log^{1/2+O(\varepsilon)} n\right) \\ Q_3(n) &= O\left(\frac{\log n}{\log\log n}\right). \end{aligned}$$

Substituting into Lemma 8, we obtain

$$U'_4(n) = O\left(\log^{2/3} s \log_s n \log^{O(1)} \log n + \log^{1/2+O(\varepsilon)} n \log_s n + \log_s n \log^2 \log n\right)$$

$$Q'_4(n) = O\left(\log s \log\log s \log\log n + \frac{\log n}{\log\log n} + \log_s n \log^2 \log n\right).$$

We set $s := 2^{\frac{\log n}{\log^3 \log n}}$ to get $U'_4(n) = O\left(\log^{2/3} n \log^{O(1)} \log n\right)$ and $Q'_4(n) = O\left(\frac{\log n}{\log\log n}\right)$. □

As we have noted, the micro-structures in Section 3 can handle reporting queries; so are the structures obtained via the van Emde transformation (see the end of the Appendix). It can be easily checked that the entire data structure can support reporting queries with extra cost $O(k)$ for k output points.

5 Higher Dimensions

We can automatically extend our result to higher constant dimensions $d \geq 3$ by using a standard degree-b range tree, which adds a $b \log_b n$ factor per dimension to the update time and a $\log_b n$ factor per dimension to the query time. With $b = \log^{\varepsilon} n$, this gives $O\left((\log n/\log\log n)^{d-1}\right)$ query time and $O\left(\log^{d-5/3+O(\varepsilon)} n\right)$ update time, improving Mortensen's result.

Alternatively, we can directly modify our micro- and macro-structures, and obtain a better update time of the form $O\left(\log^{d-2+O(1/d)} n\right)$, as we now show.

In this section, all input points and query boxes lie in d dimensions. A j-sided query ($d \leq j \leq 2d$) is for a box that projects to bounded intervals along $j-d$ coordinate axes and to half-intervals along the remaining $2d-j$ coordinate axes—the formal set of $j-d$ axes are called *double-sided*.

Definition 1. *Define a* $\mathcal{P}_{j,\ell}(s,n)$ structure *to be a dynamic data structure for j-sided orthogonal range emptiness on a set of n points in d dimensions, where the all j-sided queries have the same set of double-sided axes, and there are at most s distinct coordinate values along $d-\ell$ of the d coordinate axes—these $d-\ell$ axes are called* short*, and the remaining ℓ axes are called* long*.*

Define a $\overline{\mathcal{P}}_{j,\ell}(s,n)$ structure *to be a $\mathcal{P}_{j,\ell}(s,n)$ structure under the further restriction that all long axes are double-sided.*

5.1 Preliminaries: Van Emde Boas transformation

Lemma 4 can be immediately generalized to higher dimensions, to handle the case of one long axis.

Lemma 9. *Given a dynamic data structure for j-sided orthogonal range emptiness on $s^{2(d-1)}$ points with (amortized) update time $U_j(s^{2(d-1)})$ and query time $Q_j(s^{2(d-1)})$, there exists a $\mathcal{P}_{j,1}(s,n)$ structure with amortized update time $U_{j,1}(s,n) = O\left(U_j(s^{2(d-1)}) \log^2 \log n\right)$ and query time $Q_{j,1}(s,n) = O\left(Q_j(s^{2(d-1)}) \log\log n\right)$.*

The $\log\log n$ *factors disappear for the* $j = d$ *case.*

The last part for $j = d$ does not require van Emde Boas recursion: for dominance queries, it suffices to maintain the minimum/maximum point at each of the $O(s^{d-1})$ lines parallel to long axis.

5.2 Micro-structures: Static universe

Lemma 5 can be generalized to the following:

Lemma 10. *For* s *points in the static universe* $\left[s^{O(1)}\right]^d$ *and a given* $b \geq 2$*, there exist data structures for dynamic orthogonal range emptiness that support*

(i) updates in $O\left(\frac{b \log^d s}{w} + 1\right)$ *amortized time and* $(d+1)$*-sided queries in* $O\left(\log_b^{d-1} s\right)$ *amortized time;*

(ii) updates in $O\left(\frac{\log^{d+1} s}{w} + 1\right)$ *amortized time and* $(2d)$*-sided queries in* $O\left(\log^d s\right)$ *amortized time.*

Lemma 10(i) is established using a buffered version of a higher-dimensional range tree, with the 2-d 3-sided structure from Lemma 5(i) for base case. The bounds for (i) above are stated with a tradeoff parameter b, which follow by increasing the fan-out of the tree (e.g., see Wilkinson's paper [27] in 2-d).

5.3 Micro-structures: Dynamic universe

To make the preceding micro-structures support a dynamic universe, the simplest way is to apply monotone list labeling (Lemma 1). Since each insertion causes an amortized $O(\log s)$ number of label changes and thus deletions and reinsertions to the data structure, the amortized update time increases by a $\log s$ factor:

Lemma 11. *For* s *points in* d *dimensions and a given* $b \geq 2$*, there exist data structures for dynamic orthogonal range emptiness that support*

(i) updates in $O\left(\frac{b \log^{d+1} s}{w} + 1\right)$ *amortized time and* $(d+1)$*-sided queries in* $O\left(\log_b^{d-1} s\right)$ *amortized time;*

(ii) updates in $O\left(\frac{\log^{d+2} s}{w} + 1\right)$ *amortized time and* $(2d)$*-sided queries in* $O\left(\log^d s\right)$ *amortized time.*

For simplicity, we will not attempt to remove the extra $\log s$ factor this time. This bypasses the complications we faced in our 2-d solution for dealing with U_X cost functions.

5.4 Macro-structures: Range tree transformation I

Lemma 7 can be generalized to the following:

Lemma 12. *Let $\ell > 0$.*

(i) Given a $\mathcal{P}_{j,\ell-1}(s,n)$ structure with (amortized) update time $U_{j,\ell-1}(s,n)$ and query time $Q_{j,\ell-1}(s,n)$, there exists a $\mathcal{P}_{j,\ell}(s,n)$ structure with amortized update and query time

$$\begin{aligned} U_{j,\ell}(s,n) &= O\left(U_{j,\ell-1}(s,n)\log_s n + \log_s n \log^2\log n\right) \\ Q_{j,\ell}(s,n) &= O\left(Q_{j,\ell-1}(s,n)\log_s n + \log_s n \log^2\log n\right). \end{aligned}$$

(ii) Given a $\overline{\mathcal{P}}_{j',\ell-1}(s,n)$ structure with (amortized) update time $\overline{U}_{j',\ell-1}(s,n)$ and query time $\overline{Q}_{j',\ell-1}(s,n)$ for $j' \in \{j-1, j\}$, there exists a $\overline{\mathcal{P}}_{j,\ell}(s,n)$ structure with amortized update and query time

$$\begin{aligned} \overline{U}_{j,\ell}(s,n) &= O\left(\left(\overline{U}_{j,\ell-1}(s,n) + \overline{U}_{j-1,\ell-1}(s,n)\right)\log_s n + \log_s n \log^2\log n\right) \\ \overline{Q}_{j,\ell}(s,n) &= O\left(\overline{Q}_{j,\ell-1}(s,n) + \overline{Q}_{j-1,\ell-1}(s,n)\log_s n + \log_s n \log^2\log n\right). \end{aligned}$$

The proof is as in the proof of Lemma 7, where we divide along some long axis (which, in (ii), must also be double-sided by definition of $\overline{\mathcal{P}}_{j,\ell}$).

Combining with our preceding micro-structures and the van Emde Boas transformation, we obtain the desired update time but slightly suboptimal query time in (ii) (which we will fix later):

Theorem 3. *Given n points in a constant dimension $d \geq 3$, there exist data structures for dynamic orthogonal range emptiness that support*

(i) updates in amortized $O\left(\frac{\log^{d-1} n}{w^{1-2/(d+1)-\varepsilon}}\right)$ time and d-sided (dominance) queries in amortized $O\left(\log_w^{d-1} n\right)$ time for an arbitrarily small constant $\varepsilon > 0$;

(ii) updates in amortized $O\left(\frac{\log^{d-1} n}{w^{1-3/(d+2)}}\log^{O(1)}\log n\right)$ time and $(2d)$-sided queries in amortized $O\left(\log^{d-1} n \log\log n\right)$ time.

Proof. For (i), applying Lemma 12(i) $d-1$ times yield a structure for dominance queries with update and query time

$$\begin{aligned} U_d(n) = U_{d,d}(s,n) &= O\left(U_{d,1}(s,n)\log_s^{d-1} n + \log_s^{d-1} n \log^2\log n\right) \\ Q_d(n) = Q_{d,d}(s,n) &= O\left(Q_{d,1}(s,n)\log_s^{d-1} n + \log_s^{d-1} n \log^2\log n\right). \end{aligned}$$

By Lemmata 11(i) and 9, we have $U_{d,1}(s,n) = O(U_d(s^{O(1)})) = O\left(\frac{b\log^{d+1} s}{w}+1\right)$ and $Q_{d,1}(s,n) = O(Q_d(s^{O(1)})) = O\left(\log_b^{d-1} s\right)$. Setting $b = w^\varepsilon$ and $s = 2^{w^{1/(d+1)}}$ yields

$$\begin{aligned} U_d(n) &= O\left(b\log_s^{d-1} n\right) = O\left(\frac{\log^{d-1} n}{w^{(d-1)/(d+1)-\varepsilon}}\right) \\ Q_d(n) &= O\left(\log_b^{d-1} s \log_s^{d-1} n\right) = O\left(\log_w^{d-1} n\right). \end{aligned}$$

For (ii), applying Lemma 12(ii) repeatedly yields a structure for $(2d)$-sided queries with update and query time

$$\begin{aligned} U_{2d}(n) = \overline{U}_{2d,d}(s,n) &= O\left(\sum_{j=d+1}^{2d} \overline{U}_{j,1}(s,n)\log_s^{d-1} n + \log_s^{d-1} n \log^2\log n\right) \\ Q_{2d}(n) = \overline{Q}_{2d,d}(s,n) &= O\left(\sum_{j=d+1}^{2d} \overline{Q}_{j,1}(s,n)\log_s^{2d-j} n + \log_s^{d-1} n \log^2\log n\right). \end{aligned}$$

By Lemmata 11 and 9, we have $\overline{U}_{j,1}(s,n) = O(U_j(s^{O(1)})\log^2\log n)$, $\overline{Q}_{j,1}(s,n) = O(Q_j(s^{O(1)})\log\log n)$, $U_j(s^{O(1)}) = O\left(\frac{\log^{d+2} s}{w}+1\right)$, $Q_j(s^{O(1)}) = O\left(\log^d s\right)$ for $j \geq d+2$, and $Q_j(s^{O(1)}) = O\left(\log^{d-1} s\right)$ for $j = d+1$. Setting $s = 2^{w^{1/(d+2)}}$ yields

$$\begin{aligned} U_{2d}(n) &= O\left(\log_s^{d-1} n \log^2\log n\right) = O\left(\frac{\log^{d-1} n}{w^{(d-1)/(d+2)}}\log^2\log n\right) \\ Q_{2d}(n) &= O\left(\left(\log^{d-1} s\log_s^{d-1} n + \log^d s \log_s^{d-2} n\right)\log\log n\right) = O\left(\log^{d-1} n \log\log n\right). \end{aligned}$$

□

5.5 Macro-structures: Range tree transformation II

Lemma 8 can be generalized to the following:

Lemma 13. *Given a $\mathcal{P}_{j',\ell'}(s,n)$ structure with (amortized) update time $U_{j',\ell'}(s,n)$ and query time $Q_{j',\ell'}(s,n)$ for $(j',\ell') \in \{(j-1,\ell),(j,\ell-1)\}$ with $j > 2d-\ell$, there exists a $\mathcal{P}_{j,\ell}(s,n)$ structure with amortized update and query time*

$$\begin{aligned} U_{j,\ell}(s,n) &= O\left(\left(U_{j-1,\ell}(s,n) + U_{j,\ell-1}(s,n)\right)\log_s n + \log_s n \log^2\log n\right) \\ Q_{j,\ell}(s,n) &= O\left(Q_{j-1,\ell}(s,n) + Q_{j,\ell-1}(s,n)\log_s n + \log_s n \log^2\log n\right). \end{aligned}$$

The proof is as in the proof of Lemma 8, where we divide along some long, double-sided axis (which exists since there are ℓ long axes and $j-d$ double-sided axes and $\ell+j-d > d$).

We obtain our final result by bootstrapping:

Theorem 4. *Given n points in a constant dimension $d \geq 3$, there exist data structures for dynamic orthogonal range emptiness that support updates in amortized $O\left(\frac{\log^{d-1} n}{w^{1-3/(d+2)}} \log^{O(1)} w\right) = O\left(\log^{d-2+3/(d+2)} n \log^{O(1)} \log n\right)$ time and $(2d)$-sided queries in amortized $O\left(\log_w^{d-1} n\right) = O\left(\left(\frac{\log n}{\log \log n}\right)^{d-1}\right)$ time.*

Proof. Applying Lemma 13 repeatedly yields a structure for $(2d)$-sided queries with update and query time

$$U_{2d}(n) = U_{2d,d}(s,n) = O\left(U_{d,d}(s,n) \log_s^d n + \sum_{j=d+1}^{2d} U_{j,2d-j}(s,n) \log_s^d n + \log_s^d n \log^2 \log n\right)$$

$$Q_{2d}(n) = Q_{2d,d}(s,n) = O\left(Q_{d,d}(s,n) + \sum_{j=d+1}^{2d} Q_{j,2d-j}(s,n)\right).$$

By Theorem 3(i), we have $U_{d,d}(s,n) = U_d(n) = O\left(\frac{\log^{d-1} n}{w^{1-2/(d+1)-\varepsilon}}\right)$ and $Q_{d,d}(s,n) = Q_d(n) = O\left(\log_w^{d-1} n\right)$.

Applying Lemma 12(i) repeatedly yields

$$\begin{aligned} U_{j,2d-j}(s,n) &= O\left(U_{j,1}(s,n) \log_s^{2d-j-1} n + \log_s^{2d-j-1} n \log^2 \log n\right) \\ Q_{j,2d-j}(s,n) &= O\left(Q_{j,1}(s,n) \log_s^{2d-j-1} n + \log_s^{2d-j-1} n \log^2 \log n\right). \end{aligned}$$

By Lemma 9, we have $U_{j,1}(s,n) = O(U_j(s^{O(1)}) \log^2 \log n)$ and $Q_{j,1}(s,n) = O(Q_j(s^{O(1)}) \log \log n)$.

By Theorem 3(ii), we have $U_j(s^{O(1)}) = O\left(\frac{\log^{d-1} s}{w^{1-3/(d+2)}} \log^{O(1)} \log s\right)$ and $Q_j(s^{O(1)}) = O\left(\log^{d-1} s \log \log s\right)$.

Putting everything together, we obtain

$$\begin{aligned} U_{2d}(n) &= O\left(\left(\frac{\log^{d-1} n}{w^{1-2/(d+1)-\varepsilon}} + \frac{\log^{d-1} s}{w^{1-3/(d+2)}} \log^{O(1)} \log s\right) \log_s^{O(d)} n\right) \\ Q_{2d}(n) &= O\left(\log_w^{d-1} n + \log^{d-1} s \log_s^{d-2} n \log^2 \log n\right). \end{aligned}$$

Setting $s = 2^{\log n / \log^{d+1} w}$ yields the result. □

6 Final Remarks

We have not yet mentioned space complexity. We can trivially upper-bound the space of our data structure by n times the update time, i.e., $O\left(n \log^{2/3+o(1)} n\right)$ for the 2-d 4-sided case, which is already an improvement over Mortensen's $O\left(n \log^{7/8+\varepsilon} n\right)$ space bound. Similarly,

we obtain $O\left(n \log^{1/2+\varepsilon} n\right)$ space for our 3-sided structures, matching the space complexity of Wilkinson's structures. It might be possible to improve space further by using more bit-packing tricks, but it is not clear at all how to reduce space all the way to near linear, especially for the 4-sided case. See also the work by Nekrich [23, 24], which can achieve near linear space but require larger, super-logarithmic update time.

We hope that our ideas on micro- and macro-structures will find more applications in dynamic geometric data structures. In fact, we have recently obtained new results [11] on dynamic 2-d orthogonal point location based on a similar approach.

References

[1] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000.

[2] Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 534–543, Nov 1998.

[3] Arne Andersson and Mikkel Thorup. Dynamic ordered sets with exponential search trees. *Journal of the ACM*, 54(3):13, 2007.

[4] Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003.

[5] Lars Arge and Jeffrey Scott Vitter. Optimal external memory interval management. *SIAM Journal on Computing*, 32(6):1488–1508, 2003.

[6] Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, pages 152–164, 2002.

[7] Jon Louis Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244–251, 1979.

[8] Gerth Stølting Brodal. External memory three-sided range reporting and top-k queries with sublogarithmic updates. In *Proceedings of the 33rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 23:1–23:14, 2016.

[9] Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the 27th Annual Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011.

[10] Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pages 161–173, 2010.

[11] Timothy M. Chan and Konstantinos Tsakalidis. Dynamic planar orthogonal point location in sublogarithmic time. In *Proceedings of the 34th Annual Symposium on Computational Geometry (SoCG)*, volume 99, pages 25:1–25:15, 2018.

[12] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988.

[13] Paul F. Dietz. Maintaining order in a linked list. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 122–127, 1982.

[14] Paul F. Dietz and Daniel Dominic Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 365–372, 1987.

[15] Otfied Fries, Kurt Mehlhorn, Stefan Näher, and Athanasios K. Tsakalidis. A $\log \log n$ data structure for three-sided range queries. *Information Processing Letters*, 25(4):269–273, 1987.

[16] Tsvi Kopelowitz. On-line indexing for general alphabets via predecessor queries on subsets of an ordered list. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 283–292, 2012.

[17] Vijay Kumar and Eric J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proceedings of the 8th Annual IEEE Symposium on Parallel and Distributed Processing*, pages 169–176, 1996.

[18] George S. Lueker. A data structure for orthogonal range queries. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 28–34, 1978.

[19] Edward M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.

[20] Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(1):215–241, 1990.

[21] Christian Worm Mortensen. Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time. In *Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pages 618–627, 2003.

[22] Christian Worm Mortensen. Fully dynamic orthogonal range reporting on RAM. *SIAM Journal on Computing*, 35(6):1494–1525, 2006.

[23] Yakov Nekrich. Space efficient dynamic orthogonal range reporting. *Algorithmica*, 49(2):94–108, 2007.

[24] Yakov Nekrich. Orthogonal range searching in linear and almost-linear space. *Computational Geometry*, 42(4):342–351, 2009.

[25] Mark H. Overmars. Efficient data structures for range searching on a grid. *Journal of Algorithms*, 9(2):254–275, 1988.

[26] Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, 1977.

[27] Bryan T. Wilkinson. Amortized bounds for dynamic orthogonal range reporting. In *Proceedings of the 22th Annual European Symposium on Algorithms (ESA)*, pages 842–856, 2014.

[28] Dan E. Willard. New data structures for orthogonal range queries. *SIAM Journal on Computing*, 14(1):232–253, 1985.

[29] Dan E. Willard. Examining computational geometry, Van Emde Boas trees, and hashing from the perspective of the fusion tree. *SIAM Journal on Computing*, 29(3):1030–1049, 2000.

[30] Dan E. Willard and George S. Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM*, 32(3):597–617, 1985.

Appendix: Proof of Lemma 4 (van Emde Boas transformation)

Mortensen proved the version of Lemma 4 for a static y-universe. We give a brief redescription of the method (which is similar to van Emde Boas trees), which can deal with dynamic y-universes.

The data structure. Let S be the input point set of size at most n. Divide the plane into $O(\sqrt{n})$ horizontal slabs each with at most $2\sqrt{n}$ points of S.

1. For each slab σ, let M_σ contain the topmost and bottommost point of $S \cap \sigma$ at each x-coordinate of X. Store this set M_σ of at most $2s$ points in a structure with $U_j(s, 2s)$ update time and $Q_j(s, 2s)$ query time.
2. For each slab σ, recursively build a data structure for the remaining points in $(S \cap \sigma) \setminus M_\sigma$.
3. Let R denote the set of points in S after "rounding" down y-coordinates to align with the slab boundary lines. Recursively build a data structure for R.

In addition, for each slab σ, store the points of $S \cap \sigma$ with the same x-coordinate in a common linked list, ordered by y. Store a pointer from each y-coordinate in S to the slab containing it. The base case is when $n \leq s^2$, where we directly use the structure with $U_j(s, s^2)$ update time and $Q_j(s, s^2)$ query time.

Let $U_{j.\text{prep}}(s, n)$ denote the *amortized* preprocessing time of the above data structure, i.e., the preprocessing time divided by the number of input points. Each point contributes to a recursive data structure for $(S \cap \sigma) \setminus M_\sigma$ or for R, but not both. It follows that $U_{j.\text{prep}}(s, n) \leq U_{j.\text{prep}}(s, O(\sqrt{n})) + O(U_j(s, 2s))$, implying $U_{j.\text{prep}}(s, n) \leq O(U_j(s, 2s) \log\log n + U_j(s, s^2))$.

Updates. To insert a point q in S:

1. find the horizontal slab σ containing q (by following pointers in $O(1)$ time);
2. if q replaces another point q' as the lowest or bottommost point of $S \cap \sigma$ at q's x-coordinate, then delete q' from M_σ, insert q to M_σ, and recursively insert q' to $(S \cap \sigma) \setminus M_\sigma$;
3. else if there is no point of $S \cap \sigma$ with q's x-coordinate, then recursively insert q to R after rounding.
4. if σ contains more than $2\sqrt{n}$ points of S, split σ into two subslabs σ_1 and σ_2 with $\sqrt{n}$ points, build M_{σ_1} and M_{σ_2} with $O(\sqrt{n})$ insertions, and update R with $O(s)$ deletions and re-insertions.

Deletions are similar (except that splitting is not necessary).

Line 4 deals with rebalancing when y-universe is dynamic. Note that it is done only after $\Omega(\sqrt{n})$ updates. Thus, the amortized update time satisfies the recurrence

$$\begin{aligned} U_j(s,n) &\leq U_j(s, O(\sqrt{n})) + O(U_j(s,2s)) + \\ &\quad O\left(\frac{1}{\sqrt{n}} \cdot (\sqrt{n}\, U_{j.\mathrm{prep}}(s, O(\sqrt{n})) + s\, U_j(s, O(\sqrt{n})))\right) \\ &\leq \left(1 + O\left(\frac{s}{\sqrt{n}}\right)\right) U_j(s, O(\sqrt{n})) + O(U_j(s,2s)\log\log n + U_j(s,s^2)). \end{aligned}$$

This implies $U_j(s,n) = O(U_j(s,2s)\log^2\log n + U_j(s,s^2))$.

Updates in X. An update in X takes $U_X(s)$ time, since all the M_σ structures and base cases share the same set X of x-coordinates.

Queries. To answer a query in the point set S for rectangle q:

1. find the (at most) two horizontal slabs σ and σ' containing the top and bottom edges of q (by following pointers in $O(1)$ time);
2. if $\sigma = \sigma'$, then answer the query in M_σ, and recursively answer the query in $(S \cap \sigma) \setminus M_\sigma$;
3. else answer the query in M_σ and $M_{\sigma'}$, and recursively answer the query in R.

The query time satisfies the recurrence $Q_j(s,n) \leq Q_j(s, O(\sqrt{n})) + O(Q_j(s,2s))$, implying $Q_j(s,n) = O(Q_j(s,2s)\log\log n + Q_j(s,s^2))$. This concludes the proof of the lemma.

Remarks on reporting. The query algorithm above can be modified to handle range reporting queries. Each point is reported once, but if we are not careful, the query time for k reported points could increase by an $O(k \log\log n)$ term, because at each of the $O(\log\log n)$ levels of recursion, we may need to "decode" each reported point in R (i.e., we need to find which points in S are rounded to that point in R).

We can fix the issue by maintaining pointers to global lists (as was proposed in Mortensen's paper). For each x-coordinate in X, we store all input points with that x-value in a global linked list, ordered by y. In each set S encountered during recursion, a point p in S corresponds to a contiguous subsequence of points in the global linked list at p's x-coordinate; we store pointers from p to the first and last point in the subsequence. Each reported point can then be decoded in $O(1)$ time, and total extra cost for reporting k points is just $O(k)$.